Table of Contents

Graphics cards use

FDTD method is computationally limited by dense discretization that is necessary for its proper operation. For the standard Yee's algorithm (that is a core of FDTD) the stable discretization requires a spatial minimum of $\Delta x \leq \lambda/10$, where λ is propagating electromagnetic wave wavelength. There are some approaches in the literature to overcome these limitations, which however lead to much more complicated algorithms that cannot be simply adapted to all of the large spectrum of FDTD calculations.

As a result, both the capacity of computer memory and the processing speed is a limiting factor for large and long calculations in FDTD. This fact decreases the applicability of FDTD for many purposes. An alternative approach, that is used in Gsvit, is based on using a graphics processing unit (graphics card) to move the calculation from the computer processor (CPU) to the graphics processing unit (GPU) and to speed it up significantly, as shown in the following graph where the time necessary for the same calculation on different cards and processors is listed (note that the x axis is logarithmc).

The capabilities of graphics cards for numerical modeling have increased dramatically over the last few years. Two main producers NVIDIA and ATI have developed drivers for running calculations on their cards, and both memory and processing power of commonly used graphical cards have increased by a large factor. In Gsvit we focus on Nvidia CUDA products (from historical reasons when this was the only graphics card that could be coded in C). Generally, the available memory and computing power of GPUs increases much more rapidly than for personal computers themselves, which is also promising. Moreover, even special supercomputers based on graphical cards can be purchased now, like Nvidia Tesla systems. In these computers there are multiple graphics cards and we want to address also computing on multiple cards therefore.

To use GPU for a calculation is not straightforward, unfortunataly. We cannot simply take a conventional PC executable and run it on GPU. Both data processing and memory model is completely different for GPU and for CPU and the part of the code that should be run on GPU (called kernel) must be written to fulfill these conditions. GPU is equipped by several multiprocessors, consisting of a large number of processors. Many hundreds of threads (kernel calls) grouped in thread blocks can be processed simultaneously on GPU, which is the basis of tremendous speedup that we can achieve. Memory available on GPU can be divided into a global memory - accessible by all the multiprocessors, a shared memory - accessible by processors within one multiprocessor, and a local memory - accessible by single processor. All the memories are hardware limited (for each type of GPU differently). We refer to Nvidia CUDA developer zone for further details.